Final Project - Computer Vision

Xiangwei Shi 4614909, Benjamin Los 4301838
IN4393 Computer Vision, Delft University of Technology, The Netherlands
June 30, 2018

1. Making matches

To start this project of 3D reconstruction, it is essential to have the matched correspondences of consecutive images at the beginning. We extracted the correspondences from all the *Model Castle* images and matched the relevant correspondences between the consecutive images, which was accomplished in the first two sections of **final_project.m**.
First of all, we used the feature point descriptors provided on Brightspace. There are 19 images in the 'model_castle' folder and two different kinds of feature point descriptors, Harris and Hessian affine, for each image. Next we extracted coordinates and SIFT descriptors of both Harris and Hessian affine interest points and concatenated them for each image respectively, which is accomplished by a function, **extract_SIFT.m**. And then we matched the SIFT descriptors by using the function from **vlfeat**. After this, we could get the coordinates and indices of the matched points between two consecutive images, which were saved in the variables of **match_points** and **matches**.

2. Optimizing matches

After matching the interest points of consecutive images, we applied normalized 8-point RANSAC algorithm to select the best matched points between consecutive images and rule out the outliers of matched points. And this was accomplished in the third section of **final_project.m**.
The function, **normalRANSAC.m**, is the main function of normalized 8-point RANSAC algorithm. Before applying **normalRANSAC.m**, we defined several parameters for the algorithm. The first parameter is the number of iteration. Small value of iteration cannot guarantee that most of the outliers are ruled out, but large value of iteration wastes the time after finding the best inliers. Therefore, we set the iteration as 100. And the second of parameter is the number of corresponds, which must be at least 8. The corresponds that were randomly selected were used to estimate the fundamental matrix for the pairs of the consecutive images. The last parameter is the threshold for the RANSAC algorithm. In the normalized RANSAC algorithm, we applied the Sampson distance as a reference to judge if a pair of match agrees with a fundamental matrix. If the Sampson distance is smaller than the threshold, we consider this pair of match as inliers. Therefore, small value of the threshold could rule out many inliers, and large value of the threshold would consider some outliers as inliers. It is difficult to determine the optimal value of threshold. After applying **normalRANSAC.m**, we got coordinates and indices of the inliers of matches between the consecutive images, which were saved in the variables of **best_match_points** and **best_matches**.

3. Chaining

Since we ruled out the outliers and got the best matched points, we established a function,**create_pv_matrix.m**, in the fourth section of **final_project.m** to create point-view matrix to represent point correspondences for different camera views.
To create a point-view matrix, we needed the best matched points between consecutive images after normalized RANSAC. For the first pair of consecutive images, we simply added the indices of the best matched points to the first two rows of the point-view matrix. And next for the subsequent frame-pairs except the last frame-pair, we compared the first image of current frame-pair and the last row of point-view matrix. We added some indices of points of the first image of current frame-pair, which did not appear in the previous frame-pair, to new columns of the last row, and all the indices of points of the second image of current frame-pair to the next row. As for the last frame-pair (the last image and the first image), there were four different situations. We firstly compared the first image of current frame-pair with the last row of point-view matrix and the second image of current frame-pair with the first row of point-view matrix. Secondly, we added the indices of points to the corresponding rows of point-view matrix, of which the indices of matched points of the other image of current frame-pair appeared before. Thirdly, we moved the indices of points that appeared in the both rows of point-view matrix into the same columns. At last, we added the new appearing indices of points in new columns of the corresponding rows. The point-view matrix was saved in the

variable of **point_view_matrix**, of which the rows represented images and the columns represented points.

4. Estimate 3D points and eliminate affine ambiguity

Once we have the point-view matrix, we could use the 2D coordinates of corresponding points to estimate the 3D points, which is the function, **generate_3d_clouds.m**.
In the function, we applied affine structure from motion by using Tomasi-Kanade factorization. For starter we selected the indices of the points that appeared in three and four consecutive images. Since the rank of the measurement matrix is 3, we could use structure from motion to estimate 3D points only if there were at least 3 points in the matrix, else we returned an empty array. Based on their indices, we created the measurement matrices for three and four consecutive images. The rows of the measurement matrices represented the x and y coordinates of each image, and the columns of the matrices represented the points. And then, we applied the structure from motion to estimate the 3D points coordinates by Tomasi-Kanadefactorization. In order to eliminate the affine ambiguity of 3D points, we used Cholesky decomposition to compute the C matrix as long as the matrix L was positive definite matrix. If the L was positive definite matrix, we could eliminate the affine ambiguity, and if not we returned the 3D points without eliminating the affine ambiguity. At last, we stored the 3D points coordinates and the indices of the columns of point-view matrix related with the 3D points in the variables of **points_3d** and **points_ind**. We found that most of 3D point clouds could be eliminated affine ambiguity, but some could not because of the non-positive definite matrix. And a few 3D point clouds for four consecutive images could not be estimated due to the lack of corresponding 2D points.

5. Stiching and procrustes analysis

After we generated the 3d point clouds for each set of consecutive images we are now going to try and stitch the different point clouds together. In order to stitch two point clouds together A and B, we need to have a transfer function from the domain of B to the domain of A. To get the transfer function we use the *MATLAB* function **procrustes**. **Procrustes** needs to sets of corresponding points, it then tries to find a transfer function from the one set to the other. The important thing is that the points should correspond to each other. The points that correspond to each other are the points that are both in cloud A and B. We find these points by taking the intersection of the indexes in the **point_view_matrix** of the points. Once we found a transfer function we can transfer the points of cloud B to cloud A.
The order in which we try to combine the clouds is important because the quality of the transformation depends on the number of intersection points between the clouds. Once two clouds are combined poorly the following clouds will be combined in the same order. We therefore want strong connections between clouds and later we could add additional (weaker) points. In our implementation we initialize the original cloud with each of the partial clouds and then we go through the process of adding other clouds until we can not add any more. Finally we return the stitched cloud that has the largest size.

6. Bundle adjustment

After stiching and procrustes analysis, we managed to combine and transform all the point clouds of each image into one main view. However, there was one problem that some points had more one 3D point coordinates, where we applied bundle adjustment to remove the duplicate coordinates and refine the structure from motion.
For bundle adjustment, we needed to minimize reprojection error.

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} D(X_{ij}, P_i X_j)^2$$

We accomplish bundle adjustment in the function, **bundle_adjust.m**, in two different ways. The first method was using the built-in Matlab function, **fminsearch**, to search the minimum of the sum of the square of distances of each 3D points. This function used derivative-free method, which was nonlinear. And for each 3D points, we set the initial point as the origin of main view.
The second method for bundle adjustment was taking the mean of the coordinates of the points from different clouds that descried the same 3D points. Assuming for one 3D point, there were *n* points from different points clouds. The sum of reprojection error for one 3D points could be written below,

$$\sum_{i=1}^{m} D(X_i, P_i X)^2 = (x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2 + \cdots + (x - x_n)^2 + (y - y_n)^2 + (z - z_n)^2.$$

Since we want to minimize the sum of reprojection error and three dimensions are independent of each other, we could simplify the reprojection error for single 3D points as below,

$$\sum_{i=1}^{m} D(X_i, P_i X)^2 = D_x + D_y + D_z,$$

where $D_x = (x-x_1)^2 + \cdots + (x-x_n)^2$, $D_y = (y-y_1)^2 + \cdots + (y-y_n)^2$ and $D_z = (z-z_1)^2 + \cdots + (z-z_n)^2$. And for D_x, D_y and D_z, they are all quadratic functions. Therefore, the minimum of quadratic function lays the point in the parabola where the first-order derivative equals zero.

Take D_x as example, let the first-order derivative be equal to 0.

$$\frac{dD_x}{dt} = 2nx - 2(x_1 + \cdots + x_n) = 0$$

And we get

$$x = \frac{x_1 + \cdots + x_n}{n}.$$

Therefore, the mean of each dimension of coordinates of the points from different point clouds that described the same 3D point minimized the reprojection error of the 3D point.

7. 3D model plotting

Ploting the 3d points is easy using **plot**. The difficulty arises when we would like to have a surface of an object instead of a cloud of points. Or when we would like to have the actual colors of the points from the images. We will first discuss how we get the color for the points in section 7.1. We will then explain how we generated a surface from a bunch of 3d points in section 7.2.

7.1. Get the color of related points

After getting the refined 3D points in one main view, we needed to visualize the 3D points. In order to get a 3D surface plot with RGB colour of the related points, we created a function, **get_color.m**. Through this function, we utilized the indices of best matched points recorded in point-view matrix to get the coordinates of the related points, and then we summed up the three-channel values of the same related points from different images respectively, counted the number of appearance of the same related points and took the average of RGB values for each related point. And then we selected corresponding color for the estimated 3D points. Finally, since we applied bundle adjustment on 3D points clouds to get the refined 3D points, we also used **bundle_adjust.m** to apply bundle adjustment on RGB values and got the refined and corresponding color of the refined 3D points.

7.2. Getting the surface

Generating the surface from a point cloud is a difficult problem:

- It is unclear which points are outliers and which are inliers. When you make an outlier a part of the surface it might become a weird spike or hide other points inside the surface.
- It is unclear which points should be connected to which. Two surfaces that are close to each other might get connected.

This all is hard enough without the additional noise of imperfect stitching and imperfect 3d point estimation.
We generated our surface using the *MATLAB* function **boundary** and plotted it using **trisurf**. **Boundary** connects the outer most points to form the surface, it has a **shrink factor** that allows us to control how far the surface should move inwards in convex parts of the boundary. And **trisurf**, as a build-in Matlab function, interprets the color over each triangle based on the RGB color of the related points.
To remove outliers from the surface, we chose to apply single linkage using the function, **linkage**. This combines clusters based on the minimum distance. This means that the eventual clusters are the furthest removed from each other,

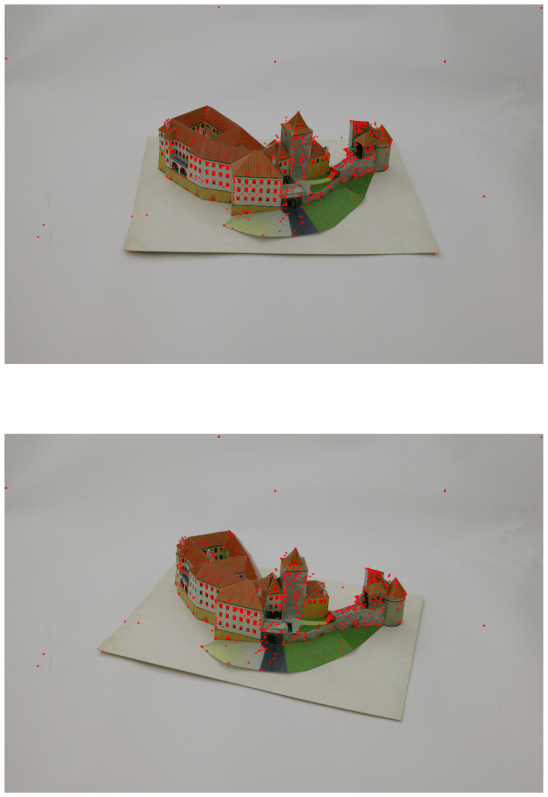

Figure 1. The matched features are depicted in red between these two images.

8. Evaluation

We will shortly evaluate a couple of steps in this project, to show that it works as expected.

8.1. Making matches

Figure 1 shows two images and the points between these images that have been matched together. We can clearly see that most matches are correct. This means that they are on the same spot on the model in both images. A couple matches are incorrect, that's why we need to optimize these matches.

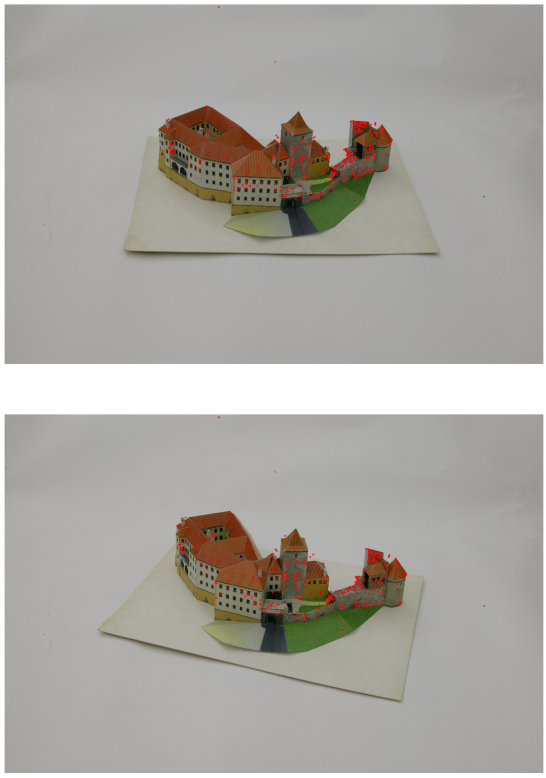

Figure 2. The matched features of figure 1 have been optimized and thereby reduced.

8.2. Optimizing matches

By using `RANSAC`, we optimize and reduce the number of features. These new features are depicted in figure 2. As shown there a far fewer points. however, most important, there are also far fewer mistakes in the matches that are made between these images.

8.3. Estimate 3D points

Now that we have the matches between the images we need to estimate their 3d location. This is depicted in figure 3. You can see that the first and second image has the same points in the same configuration. In the third image you can see that there is 3d structure within these points.

8.4. Stitching

To show that stitching works we chose to depict one of the intermediate steps during the stitching of a point cloud. We used the teddy bear as an example because the progress is better visible with this model. the green dots are the intersection and are used to create a transfer function from the new cloud to the cumulative cloud. We can see that the new (blue) point fit rather well with the old (red) points. This means that the transfer function is quite accurate.

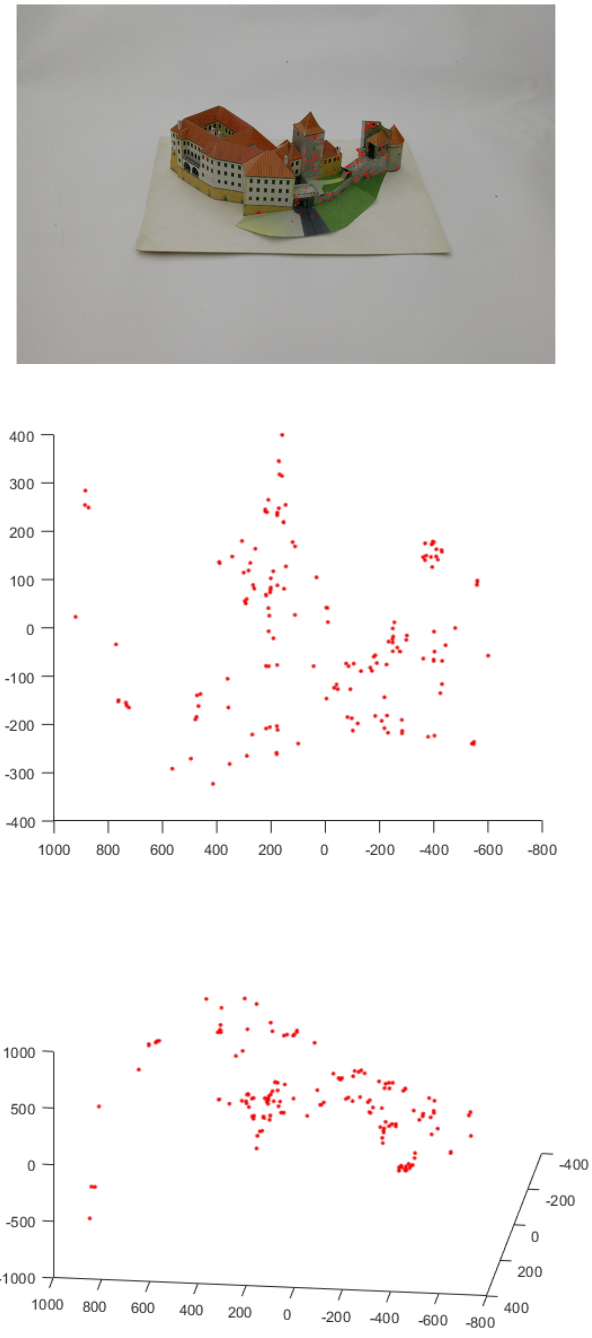

Figure 3. The first image shows the points that are matched between the images 1, 2 and 3. The second image shows the 3d estimated point cloud of these points from the same perspective. The third image shows the same point cloud but then viewed from above.

8.5. 3D model plotting

As depicted in figure 5, we can see that although the model is quite rough it is clearly depicting that part of the physical model. We can therefore see that we are successfully able

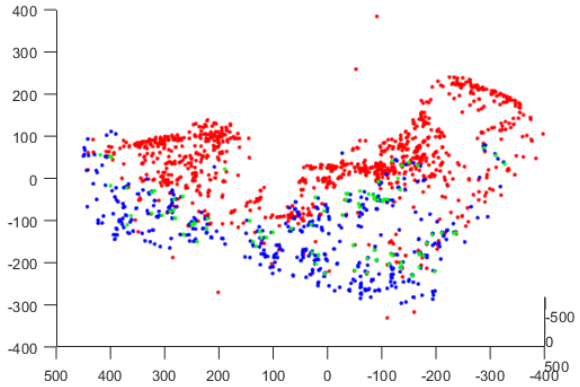

Figure 4. Stitching of the teddy bear model. In this figure one of the combining of clouds during stitching is depicted, between the cumulative cloud and a new cloud. The red dots are the points that are only in the cumulative cloud. The green dots are the points both in the cumulative cloud and the new cloud. The blue dots are the points only in the new point cloud.

Figure 5. Left is a plot of our model with the following parameters (*RANSAC iterations = 50, RANSAC threshold = 200, consecutive images = [3,4,5] , number of intersection points for stitching > 9, number of clusters = 40*). Right is a part of the picture of the corresponding part of the model.

to: find features; match them together; generate 3d points; stitch them together; apply bundle adjust; remove additional noise/outliers; plot the model with color and a surface.
We can of course also see that this is only a subset of the entire model. This indicates that we are not able to generate the remaining part of the model. Either due to matching points, generating 3d points, stitching them together, or distinguish it from the noise.